FUNCTION

LABORATÓRIO DE BANCO DE DADOS

VANESSA BORGES

Funções

- Funções são subprogramas que têm por objetivo **executar uma ação específica**

- Importante recurso suportado pelos SGBDs há vários anos
  - Conjunto de código SQL armazenados no servidor do SGBD que podem ser invocados a qualquer momento por meio de um comando especial

Funções – Vantagens

- Vantagens
  - **Centralização**
    - Em casos onde o banco de dados é acessado por **diversas aplicações desenvolvidas** em diferentes linguagens e por diferentes equipes, com o uso de Funções as alterações e regras são mantidos em um único local
  - **Segurança**
    - As vezes é necessário utilizar alguns recursos de segurança para evitar que usuários e desenvolvedores cometam erros
  - **Abrangência da linguagem SQL**
    - Como é possível utilizar comandos de controle de fluxo e iterações estendeu-se o poder da linguagem SQL

Funções x Procedimentos

- Semelhantes por definição, funções e procedimentos são estruturas definidas para executar um conjunto de instruções utilizando variáveis de entrada.
  - **São armazenadas no SGBD e podem ser invocadas a qualquer momento**

PROCEDURE	FUNCTION
Executa um conjunto de instruções	Calcula resultados usando as entradas fornecidas
Não possui retorno	Utiliza RETURNS para retornar valores
Utiliza variáveis de saída (OUT) para retornar valores	Pode também utilizar variáveis de saída para retornar valores
Não pode ser chamado no SELECT	Pode ser chamada na instrução SELECT
Não pode ser chamada por uma função	Pode ser chamado por um procedimento

Funções PL/PGSQL

- **PL/PGSQL (*Procedural Language extensions to SQL*)**
  - Linguagem mais utilizada
  - Mais recursos
    - Utiliza variáveis e estruturas de comando
    - Pode executar ações SQL

- **Recursos procedurais**
  - Condicionais
    - IF
    - CASE
  - Loop
    - LOOP
    - FOR
    - WHILE
  - Retornos

Funções PL/PGSQL – Expressões

- As expressões em uma função podem ser aritméticas, booleanas, relacionais ou um comando SQL.
- Aritmética

```
Resultado := (valor_a + valor_b)/10;
```

- Boolean

```
Resultado_b := condicao1 AND condicao2;
valor_a >= valor_b;
```

- Relacional

```
INSERT INTO cliente (nome_cliente) VALUES (‘Rodrigo’);
```

Funções PL/PGSQL – Atribuição

- **Atribui a uma variável ou campo um valor**, resultado de uma expressão, de uma variável ou de uma constante

- Sintaxe:

```
<identificador> := <expressão>;
```

- Exemplo:

```
resultado := ((valor_a + valor_b)*50)/7;
```

Funções PL/PGSQL – Parâmetros FUNCTION

```
CREATE [OR REPLACE] FUNCTION nome_função (<lista de parâmetros>,...)
RETURNS <Retorno do tipo de dado> AS $$

[DECLARE
    <área de declaração>]
BEGIN
    <área de comandos>

END;

$$ LANGUAGE <nome da linguagem>;
```

{IN | OUT | INOUT} <nome> <tipo de dado>

Onde:

- Modo dos parâmetros
  - IN (de entrada; passagem por valor, default)
  - OUT (de saída; passagem por referência)
  - IN OUT (de entrada/saída; passagem por referência)
- Tipo de dado: INTEGER, VARCHAR, etc

- **As funções não deve possuir o nome de outra função já existente com os mesmos tipos de argumentos de entrada presentes no mesmo esquema.**
  - O que difere as funções são os argumentos de tipos diferentes, pois estes podem compartilhar um mesmo nome, o que o torna uma função de sobrecarga.

Funções PL/PGSQL – Declaração FUNCTION

```
CREATE [OR REPLACE] FUNCTION nome_função (<lista de parâmetros>,...)
RETURNS <Retorno do tipo de dado> AS $$

[DECLARE
        name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := | = } expression
];

BEGIN
    <área de comandos>

END;

$$ LANGUAGE <nome da linguagem>;
```

Exemplo:

```
quantity integer DEFAULT 32;
url varchar := 'http://mysite.com';
user_id CONSTANT integer := 10;
```

https://www.postgresql.org/docs/current/plpgsql-declarations.html#PLPGSQL-DECLARATION-COLLATION

Funções PL/PGSQL – Comandos FUNCTION

```
CREATE [OR REPLACE] FUNCTION nome_função (<lista de parâmetros>,...)
RETURNS <Retorno do tipo de dado> AS $$

[DECLARE
    <área de declaração>]

BEGIN
    <área de comandos>

END;

$$ LANGUAGE <nome da linguagem>;
```

BEGIN e END – delimitam um bloco dentro da função. No mínimo, uma função deve ter um bloco declarado. É possível declarar um bloco dentro de outro bloco (sub-blocos)

Funções em SQL – Parâmetros de funções

Existem duas formas de utilizar um alias no parâmetro

```
-- Passagem de parâmetro explícita
CREATE FUNCTION taxa_venda(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE PLPGSQL;

-- Utilizando renomeamento do parâmetro por posição $1, $2,..., $n
CREATE FUNCTION taxa_venda(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE PLPGSQL;
```

Funções PL/PGSQL – Mensagem RAISE NOTICE

- Apresenta uma mensagens (depuração) enquanto estiver rodando a função
  - **Apenas mostra uma mensagem sem interferir no fluxo normal da função**

```
CREATE OR REPLACE FUNCTION tempo_usuario(idusuario int, comentariousuario text)
RETURNS VOID AS $$
DECLARE
    tempo timestamp := now();
BEGIN
    RAISE NOTICE 'Atualizando registro id = %',idusuario;
    UPDATE usuario SET ultima_atualizacao = tempo, comentario = comentariousuario
    WHERE id = idusuario;
END;
$$ LANGUAGE PLPGSQL;
```

Funções PL/PGSQL – Exceção RAISE EXCEPTION

- **Dispara uma exceção e para a execução da função**

```
CREATE FUNCTION tempo_usuario(idusuario int, comentariousuario text) RETURNS VOID AS $$
DECLARE
    tempo timestamp := now();
BEGIN
    IF $1=3 THEN
        RAISE EXCEPTION 'Finalizando o fluxo';
    END IF;
    UPDATE usuario SET ultima_atualizacao = tempo, comentario = $2
    WHERE usuario.id = $1;
END;
$$ LANGUAGE PLPGSQL;
```

Funções PL/PGSQL – Sobrecarga

- Permite que duas funções tenham o mesmo nome desde essas funções possuam números diferentes de argumentos

- Exemplo:

```
CREATE FUNCTION avgsalario() RETURN NUMERIC AS $$ ...

CREATE FUNCTION avgsalario(funcionario) RETURN NUMERIC AS $$ ...
```

Estrutura de Controle de Fluxo

- Como em qualquer outra linguagem procedural, a PL/pgSQL possui estruturas de controle para manipular os dados.

- As estruturas de controle são:
  - Condicionais

```
IF ... THEN ... END IF
IF ... THEN ... ELSE ... END IF
IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF
```

  - Loops

https://www.postgresql.org/docs/current/plpgsql-control-structures.html

Estrutura de Controle de Fluxo

- Como em qualquer outra linguagem procedural, a PL/pgSQL possui estruturas de controle para manipular os dados.

- As estruturas de controle são:
  - Condicionais

```
IF ... THEN ... END IF
IF ... THEN ... ELSE ... END IF
IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF


CASE ... WHEN ... THEN ... ELSE ... END CASE
CASE WHEN ... THEN ... ELSE ... END CASE
```

https://www.postgresql.org/docs/current/plpgsql-control-structures.html

Estrutura de Controle de Fluxo
IF ... THEN ... ELSE

- IF / THEN / ELSE

```
CREATE OR REPLACE FUNCTION tipoSalario(salario numeric) RETURNS text AS $$
BEGIN
    IF salario < 5000 THEN
        RETURN 'Estagiário';
    ELSIF salario BETWEEN 5000 AND 20000 THEN
        RETURN 'Auxiliar';
    ELSIF salario BETWEEN 20000 AND 30000 THEN
        RETURN 'Efetivo';
    ELSE
        RETURN 'Não parametrizado';
    END IF;
END;
$$ LANGUAGE PLPGSQL;

-- Chamada da função
SELECT tipoSalario(10000);
SELECT pnome, salario, tipoSalario(salario) FROM funcionario;
```

Estrutura de Controle de Fluxo

- Loops
  - LOOP
    - Repetido indefinidamente até ser encerrado por uma instrução EXIT ou RETURN
  - FOR
    - Repete a partir de uma faixa de valores
  - WHILE
    - Repete enquanto a expressão retornar verdadeira
      - A expressão é verificada antes de cada entrada no corpo do loop

Estrutura de Controle de Fluxo
LOOP

- LOOP
  - Repetido indefinidamente até ser encerrado por uma instrução EXIT ou RETURN

```
-- loop simples
[<label>]
LOOP
    statements
    [EXIT [<label>] [ WHEN boolean-expression ];]
    [RETURN;]
END LOOP [<label>];
```

Estrutura de Controle de Fluxo
LOOP

- LOOP
  - Exemplo

```
LOOP
    -- algumas validações
    IF count > 0 THEN
        EXIT;  -- finaliza o loop
    END IF;
END LOOP;
```

```
LOOP
    -- algumas validações
    EXIT WHEN count > 0;  -- finaliza o loop
END LOOP;
```

Estrutura de Controle de Fluxo WHILE

- WHILE – é um laço que apresenta uma condição de parada na sua estrutura. Ou seja, avalia uma condição e enquanto ela for verdadeira o laço continua.

```
-- WHILE
[<label>]
WHILE boolean-expression LOOP
    statements
END LOOP [<label>];
```

- Exemplo:

```
WHILE count > 0 LOOP
    -- realiza algumas validações aqui
END LOOP;
```

Estrutura de Controle de Fluxo FOR

- FOR – é um laço de controle de fluxo com a seguinte **sintaxe**:

```
-- FOR
[<label>]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP
    statements
END LOOP [<label>];
```

- Onde:
  - `name` – é do tipo inteira é existe somente dentro do laço
  - `expression` – define os valores inicial e final do laço

Estrutura de Controle de Fluxo FOR

- FOR
  - Exemplo

```
FOR i IN 1..45 LOOP
    -- dessa forma, teremos que os valores serão trazidos em ordem crescente com    o Loop FOR
END LOOP;
```

```
FOR i IN REVERSE 45..1 LOOP
    -- dessa forma, teremos que os valores serão trazidos em ordem decrescente  com o Loop FOR
END LOOP;
```

Estrutura de Controle de Fluxo
FOR QUERY

- Laço iterando a partir do resultado de uma consulta
  - RECORD é uma estrutura genérica que assume a estrutura da linha no momento da atribuição do valor

```
CREATE OR REPLACE FUNCTION loop_select() RETURNS VOID AS $$
DECLARE registro RECORD;
BEGIN
    FOR registro IN  SELECT * FROM funcionario LOOP
        RAISE NOTICE 'Valores: %, %, %, %, %, %', registro.pnome, registro.cpf,
     registro.sexo, registro.salario, registro.cpf_supervisor, registro.dnr;
    END LOOP;
END;
$$ LANGUAGE PLPGSQL;

-- Chamada da função
SELECT loop_select() ;
```

Controle de Fluxo
Tipos de retorno - RETURN

- Existem dois comandos disponíveis que permitem retornar dados de uma função
  - RETURN
    - Termina a função e retorna o valor da expressão para o chamador. Usado que não retornam um conjunto.

```
-- Cria a função que retorna a soma dois números
CREATE OR REPLACE FUNCTION calculos_matematicos_soma(x int,y int) RETURNS int AS $$
DECLARE
    soma int;
BEGIN
    soma := x + y;
    RETURN soma;
END;
$$ LANGUAGE PLPGSQL;

 -- Executa a função
 SELECT calculos_matematicos_soma(1,2);
```

https://www.postgresql.org/docs/current/plpgsql-declarations.html

Controle de Fluxo
Tipos de retorno – RETURN NEXT

- RETURN NEXT
  - RETURN NEXT pode ser usado com tipos de dados escalares e compostos; com um tipo de resultado composto, uma “tabela” inteira de resultados será retornada.
    - **SETOF** que indica que a função irá retornar um conjunto de itens.

```
CREATE TEMP TABLE funcionario (usuarioid INT, supervisorid INT, nome TEXT);

CREATE OR REPLACE FUNCTION getfuncionario() RETURNS SETOF funcionario AS $$
DECLARE
    registro funcionario%rowtype;
BEGIN
    FOR registro IN SELECT * FROM funcionario WHERE usuarioid > 0 LOOP
        -- executa alguma validação aqui
        RETURN NEXT registro; -- retorna a linha corrente do registro
    END LOOP;
    RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM getfuncionario();
```

Controle de Fluxo
Tipos de retorno – RETURN QUERY

- RETURN QUERY
  - RETURN QUERY anexa os resultados da execução de uma consulta ao conjunto de resultados da função.

```
CREATE OR REPLACE FUNCTION getfuncionario() RETURNS SETOF funcionario AS $$
BEGIN
    RETURN QUERY SELECT * FROM funcionario WHERE usuarioid > 0;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Sem registros';
    END IF;
    RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM getfuncionario();
```

Controle de Fluxo
Tipos de retorno

- RETURN NEXT e RETURN QUERY
  - Quando uma função é declarada para retornar algum tipo de SETOF, os itens individuais a serem retornados são especificados por uma sequência de comandos RETURN NEXT ou RETURN QUERY e, em seguida, um comando RETURN final sem nenhum argumento é usado para indicar que a função terminou de ser executada.
  - RETURN NEXT pode ser usado com tipos de dados escalares e compostos; com um tipo de resultado composto, uma “tabela” inteira de resultados será retornada.
  - RETURN QUERY anexa os resultados da execução de uma consulta ao conjunto de resultados da função.
  - RETURN NEXT e RETURN QUERY não são o retorno efetivamente da função - eles simplesmente acrescentam zero ou mais linhas ao conjunto de resultados da função. A execução então continua com a próxima instrução na função PL pgSQL. Conforme os comandos RETURN NEXT ou RETURN QUERY sucessivos são executados, o conjunto de resultados é construído. Um RETURN final, que não deve ter nenhum argumento, faz com que o controle saia da função (ou você pode apenas deixar o controle chegar ao final da função).

Controle de Fluxo
Tipos de retorno – RETURN QUERY - RECORD

- RECORD pode ser utilizado como um tipo de retorno genérico

```
CREATE TEMP TABLE funcionario (usuarioid INT, supervisorid INT, nome TEXT);

CREATE OR REPLACE FUNCTION getfuncionario() RETURNS SETOF RECORD AS $$
BEGIN
    RETURN QUERY SELECT usuarioid, nome FROM funcionario WHERE usuarioid > 0;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Sem registros';
    END IF;
    RETURN;
END;
$$ LANGUAGE PLPGSQL;

-- É necessário especificar as colunas que serão recuperadas
SELECT * FROM getfuncionario() AS (usuarioid int, nome text);
```

Foram listadas colunas específicas

- Onde:
  - **SETOF** que indica que a função irá retornar um conjunto de itens
  - **RETURN QUERY** foi utilizado para ”capturar os dados”
  - **RETURN** (sozinho) irá realizar o retorno do conjunto de dados

Funções – Exceção DROP

- Apaga uma função

```
DROP FUNCTION <nome da função> (<lista de parâmetros>);
```